Designing Data-Intensive

Applications

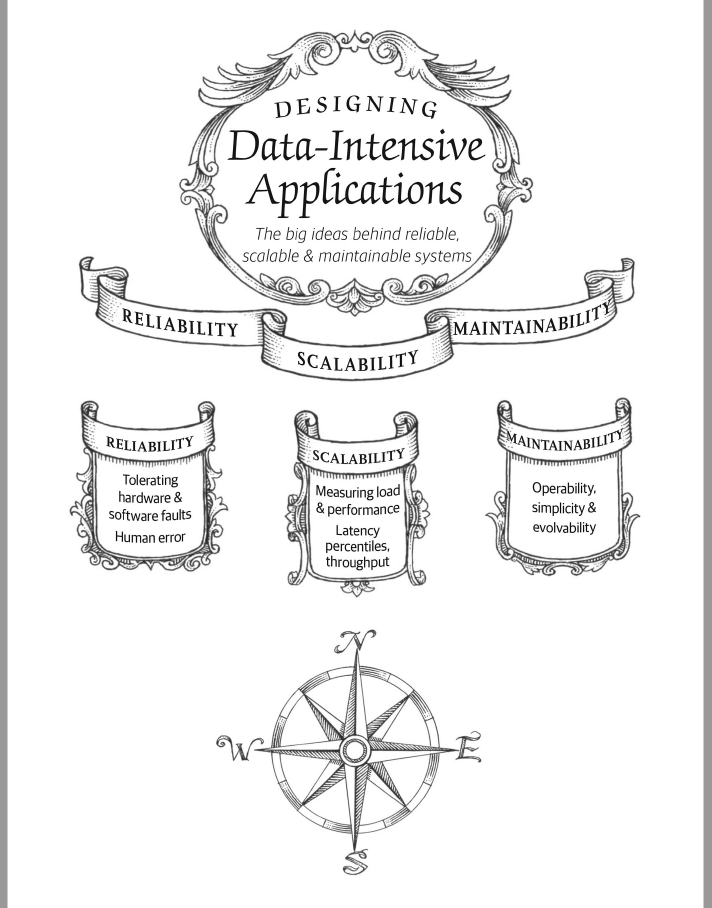
***The Big Ideas Behind Reliable, Scalable,***

***and Maintainable Systems***

* Data is at the center of many challenges in system design today
* Issue need to figure out such as scalability, consistency, reliability, efficiency and maintanbility
* Have an overwhelming variety of tools (relational databases, NoSQL datastores, stream or batch processor and message brokers)

Preface

* NoSQL! Big Data! Web-scale! Sharding! Eventual consistency! ACID! CAP theorem! Cloud services! MapReduce! Real-time!
* Many interesting developments in database, distributed system and in the way we build application on top of them
* There are various driving forces
  + Internet companies such as Google, Yahoo, amazon, facebook …handling huge volume of traffic, forcing to create tools tant enable handle such scale
  + Business need to be agile, test hypotheses cheaply and respond quickly to new market by keeping development cycle short and data models flexible
  + Free and open source software f has become very successful and is now preferred to commercial or bespoke in house software in many environement
  + CPU clock speed are barely increasing , but multi core processor are standar and networks are getting faster. The means parallelism is going to increase
  + Can work and buld systems that are distributed across many machines and even multiple geographic regions (IaaS as Amazon web service)
  + Service expected to be highly available; extended downtime
* If data is the primary challenge (the quantity of data , the complexcity or speed at changing
* message queues, caches, search indexes, frameworks for batch and stream processing,
* On this journey, we will try to find useful ways of thinking about data systems—not just how they work, but also why they work that way, and what questions we need to ask.
* You want to learn how to make data systems scalable, for example, to support web or mobile apps with millions of users.
* You need to make applications highly available (minimizing downtime) and operationally robust.
* You are looking for ways of making systems easier to maintain in the long run, even as they grow and as requirements and technologies change.



CHAPTER 1

Reliable, Scalable, and

Maintainable Applications

The Internet was done so well that most people think of it as a natural resource like the

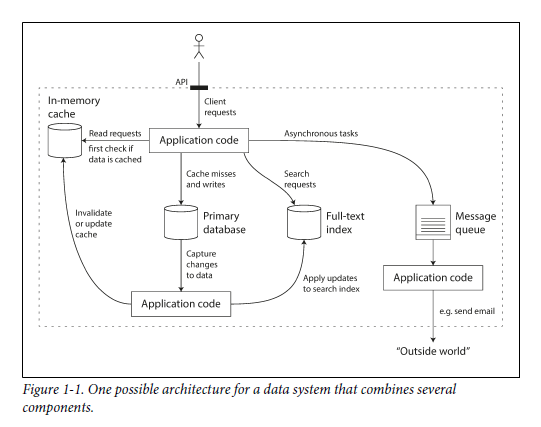
Pacific Ocean, rather than something that was man-made. When was the last time a technology with a scale like that was so error-free?

—Alan Kay, in interview with Dr Dobb’s Journal (2012)

* Many application today is data intensive as to opposed to compute intensive
  + Raw CPU paower is rarely a limitiong factor for these applications
  + Biggest problem is the amount of data , the complexity of data and the speed at which it is changing
* Many applications need to
  + Store data so that they or another application can find it again later (database)
  + Remember the result of an expensive operation, to speed up realds (caches)
  + Allow user to search data by keyword or filter it in various way (search indexes)
  + Send message to another process to be handled asynchronously (strema processing)
  + Periodically crunch a large amount of accumulated data (batch processing)
* In reality there is
  + Many databases system with differemnt characteristic(applications have different requirements)
  + Various approaches to caching
  + Several way to building search indexes
* When building an application we need to figure out which tools and approaches are most appropriate for the task at hand

Thinking About Data Systems

* There is different categories of tools (database, queues, caches etc),
  + they can also have some superficial similarity (ex database and message queue: both store data for some time)
  + they have also different access patterns, which means different performance characteristics and thus very different implementations
* Why lump them all together under an umbrella term like data systeme
  + Many tools for storage and processing have emerged in recent years
  + They are optimized for a variety of different use cases and they no longer neatly fit into traditional categories
    - Ex: data storage that are used as message queue(Redis) and there are message queues with database like durability guarantees (Apache Kafka) the boundaries between the categories are becoming blurred
  + Application with such demanding and wide-ranging requirements that a single tool can not longer meet all of its data processing and storage needs
  + Preferred brok the work that can ve perfomed efficiently on a single tool and those different tools are stitched together using application code
    - Ex application managed caching layer (memcahced) + full text search serve (elastics search) + main database …. Application code responsibility to keep those caches and indexes in sync



* Application programming interface (API) or service’s interface combine several tools usually hides from clients
  + Caches will be correctly invalidated or updated for consistent results
* Questions to taking in account
  + How do I ensure that data remain correct and complete, even thing go wrong internally
  + How do I provide consistenly good performance to clients, even if part of the system are degraded
  + How do I scala to handle an increase in load
  + What does a good API for the service look like
* Many factors influence the design of a data system
  + Skills and experience of the people involved
  + Legacy system dependencies
  + Time scale for delivery
  + Tolerance admitted of differents kind of risk
  + Regulatory constraints
* 3 concerns that are important in most softwate systems:
  + Reliability: the system should continue to work correctly (performing the correct function at the desired level of performance) (hardware / software // human error)
  + Scalability: As the system grows (data volume, traffic volume, or complexity), there should be reasonable ways of dealing with that growth
  + Maintainability: over time, many different people will work on the system and they should be able to work on it productively

Reliability

* For software expectation include
  + The application performs the functions that the user expected
  + It can tolerate user making mistakes or using the software in unexpected ways
  + Its performance is good enough for the required use case under the expected load and data volume
  + The system prevents any unauthorized access and abuse
* Meaning continue working correctly even things go wrong
* Faults (name given to things can go wrong in an application)
* Fault tolerant – resilient are systems which anticipate faults and can cope with them
* Make sense to talk about tolerating certain types of faults
* Faults # failure
  + Faults: is when a component of the system deviationg from it spec
  + Failure: is when system as a whole stops providing the required service to the user
* Seems impossible to reduce faults to 0 so rather try to prevent faults from causing failure
* It can make sense to increase the rate of faults bt triggering them deliberately (Ex Randomly killing individual processes) by doing that we ensure that the fault tolerance machinery is continually exercised and tested, which can increase yout confidence that faults will be handled correctly when they occur naturally (Netflix Chaos Monkey)
* Tolerating faults preferable to preventing faults .. but there is some king of issue that we should prevent (ex security issue )

Hardware Faults

* Many factor scan caused issue on hardware layer
  + Hard disks crash
  + RAM becomes faulty
  + Power grid has a blackout
  + Unplungs the wrong network cable
* Hard disk are reported as having a mean time to failure (MTTF) about 10 to 50 years [5,6]. Thus storage cluster with 10 000 we should expect on average one disck to die per day
* One approach it to add redundancy, so that the redundant component can take its place while the broken component is replaced
* Data volumes and applications computiong demands have increased , which proportionally increases the rate of hardware faults
* Hence there is a move toward systems that can tolerate the loss of entire machines by fault tolerance technique in preference or in addition to hardware redundancy
* Rolling upgrade : failure on one node at a time without downtime of the entire system

Software Errors

* Usually hardware faults being random and independent from each other, can have the same cause (temperature
* The can have systematic error within the system, harder to anticipate because they are uncorrelated hardware faults
  + A software bug that cause every instance of an application server to crash
  + A runaway process that uses up some shared resource – CPU time, memory, network bandwidth
  + A service that the system depends on that slows down, become unresponsive or starts returning corrupted responses
  + Cascading failures, where a small fault in one component triggers a fault in another component
* Bugs caused these kings of software faults often lie dormant for long time until they are triggered by an unusual set of circumstances
* There is no quick solution for these kind of failure
  + Carefully thinking about assumption and interaction in the systems
  + Thorough testing
  + Process isolation
  + Allowing processes to crash and restart
  + Measuring, monitoring and analyzing system behavior in production
  + A system can check itself guarantee it should provide and raise alert if a necessary

Human Errors

* Human design and build software and the operator who keep the system running is also human
* How do we make our systems reliable in spite of unreliable humans?
  + Design system in a way that minimizer opportunities for error
  + Decouple the places where people make the most mistakes from the places where they can cause failures. Provide fully featured non-production sandbox
  + Test thoroughly at all levels, from not tests to whole system integration tests and manual tests (automated testing )
* Allow quick and easy recovery from human errors, to minimize the impact in the case of a failure Ex make it fast to rollback configuration change, or roll out new code gradually , provide tools to reompute data (12)

How Important Is Reliability?

* Bugs in business appplications cause lost productivity , can haave huge cost in terms of lost revenue and damage reputation

Scalability

* Even if a system is working reliably today, that doesn’t mean it will necessarily work reliably in the future ( degradation is increased load)
* Scalability is the term used to describe system ability to cope with increased load.
* Scalability means considering questions like “
  + if the system grows in a particular way what are our options for coping with the growth
  + how can we add computing resources to handle the additional load

Describing Load

* load can be described with a few numbers which we call load parameters